Power Management on an ARM11 platform

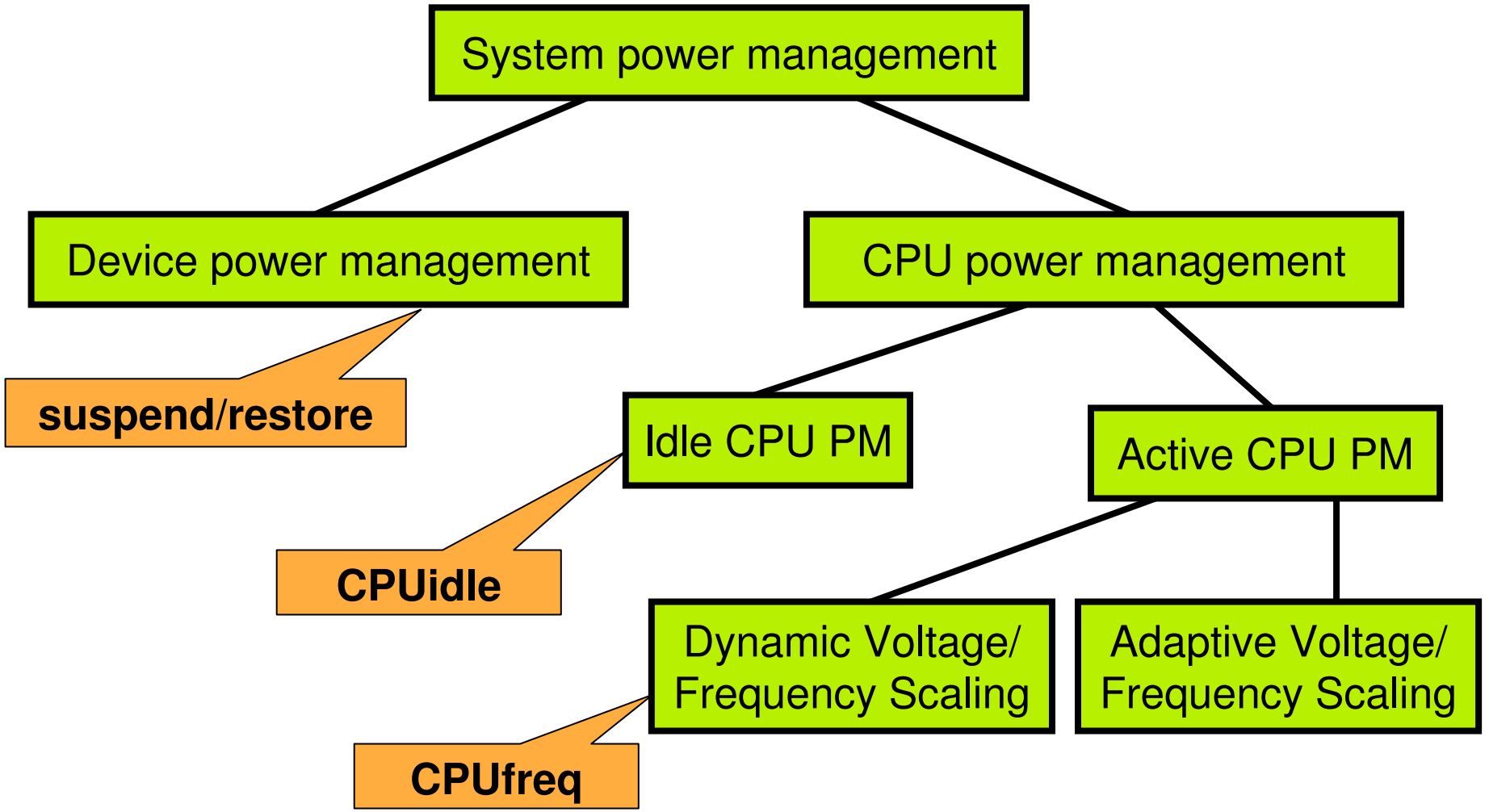
Mischa Jonker

November, 7, 2008

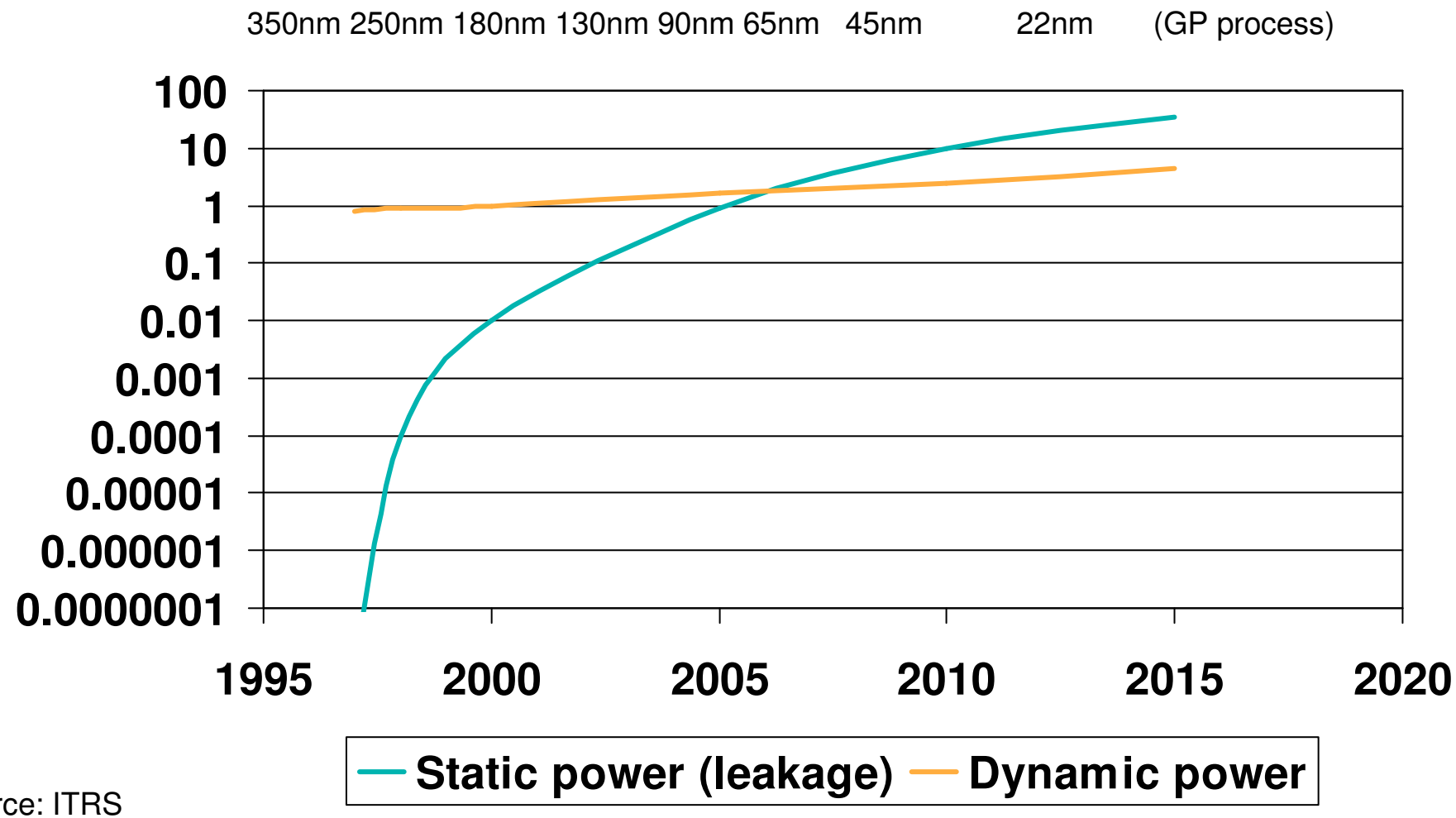
Overview

- Introduction
- Architecture of the NXP ARM1176 platform
- ARM11 power modes
- Switching between power modes
- Benchmark results
- Conclusion + Future work

Ways to save power in Linux

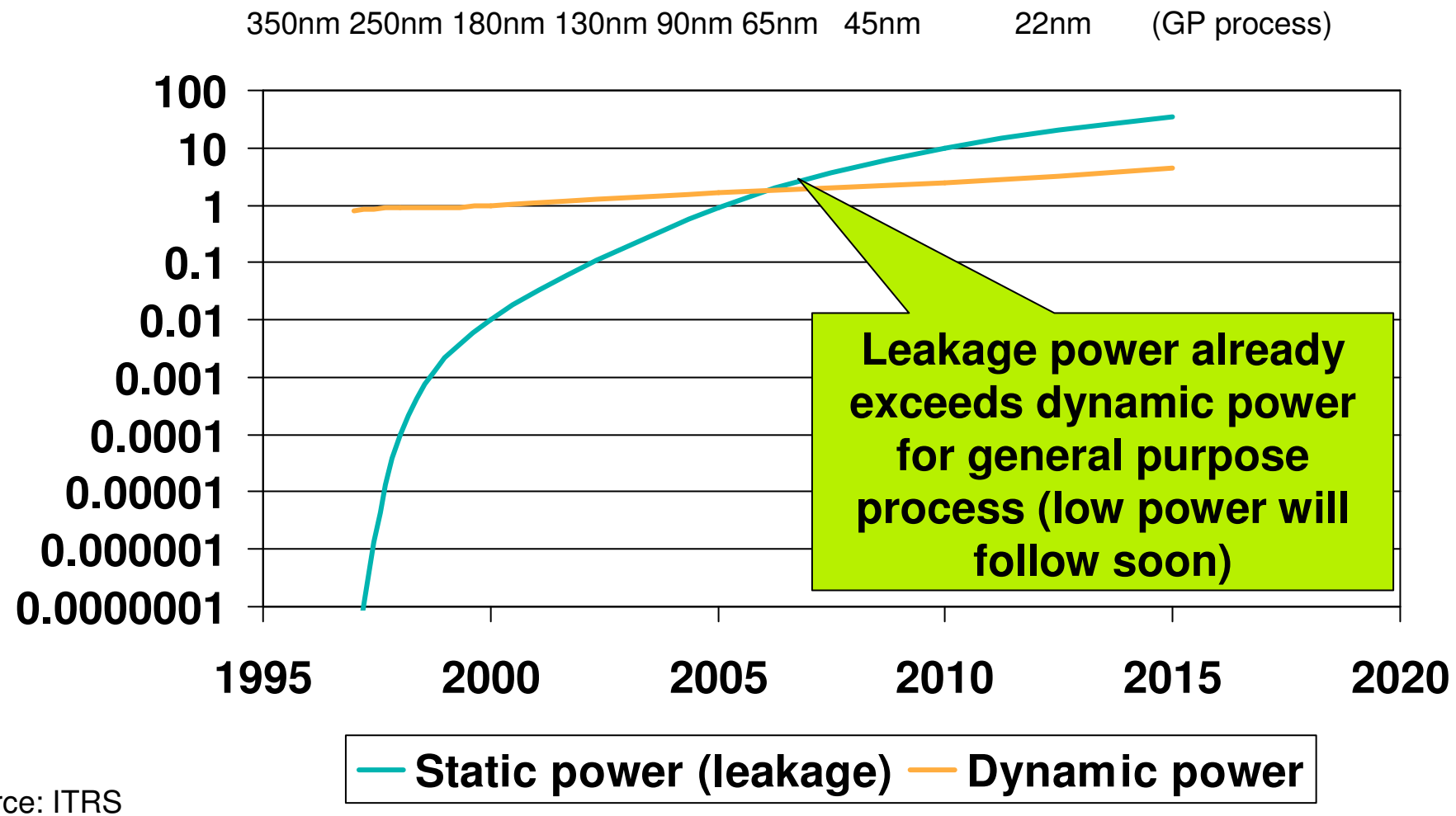


CMOS power trends



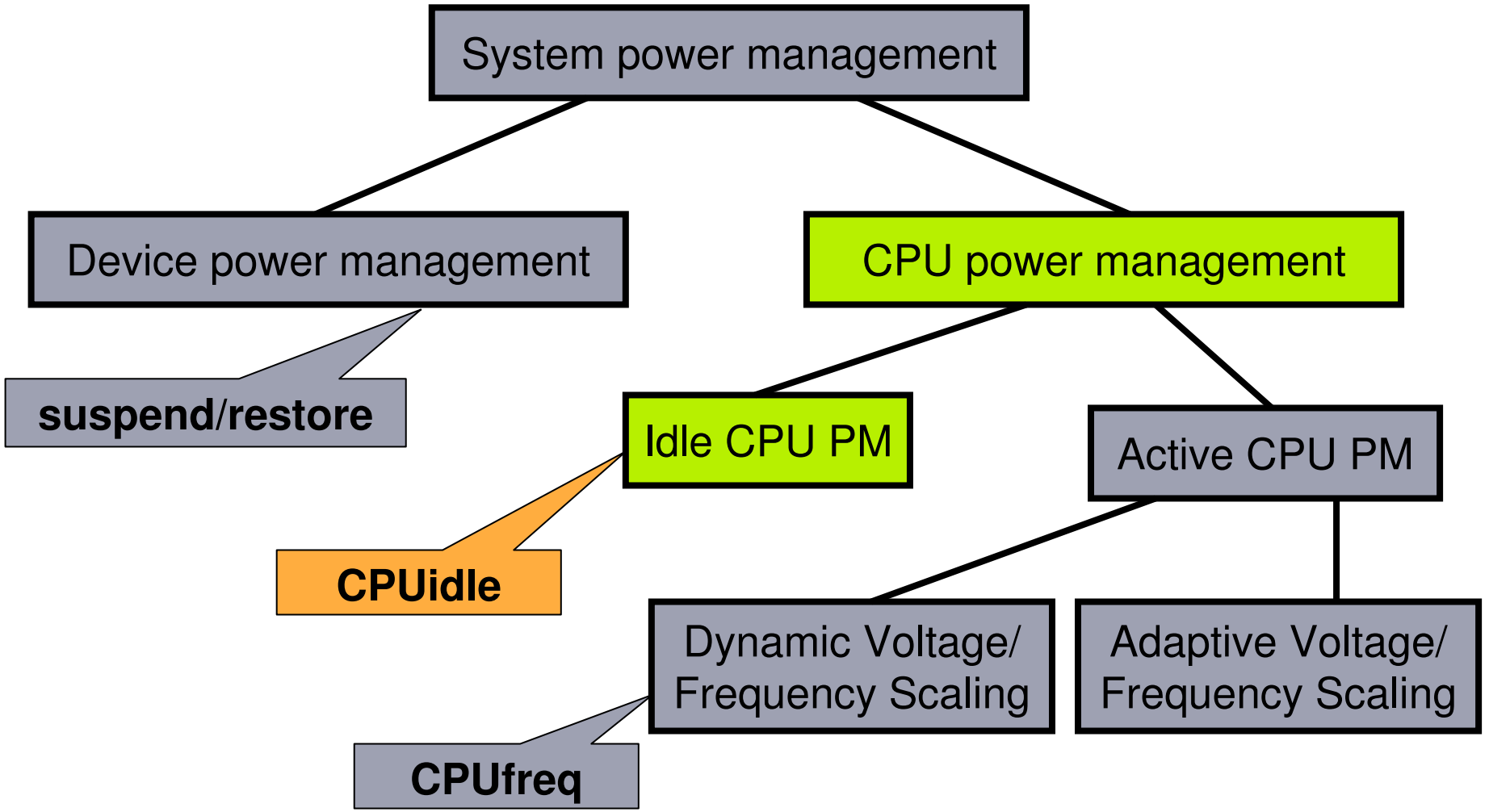
source: ITRS

CMOS power trends

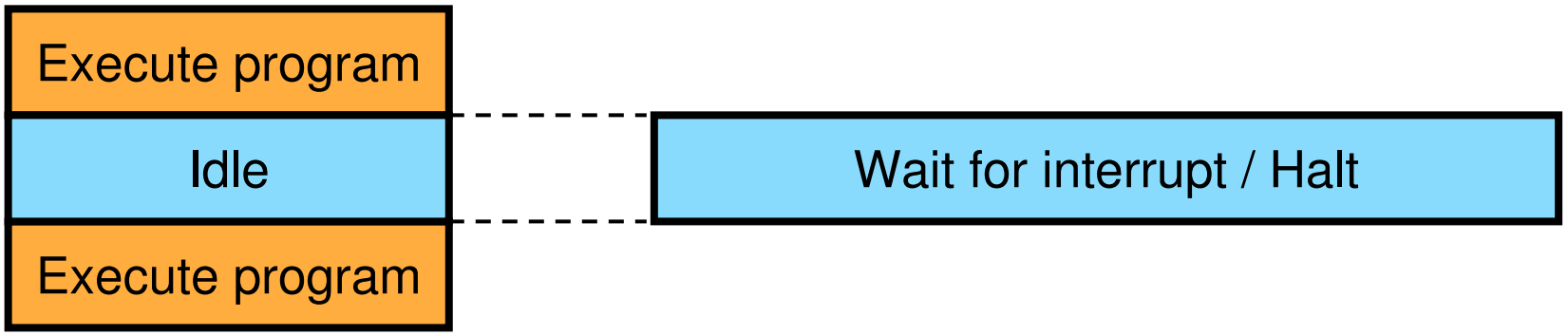


source: ITRS

Ways to save power in Linux

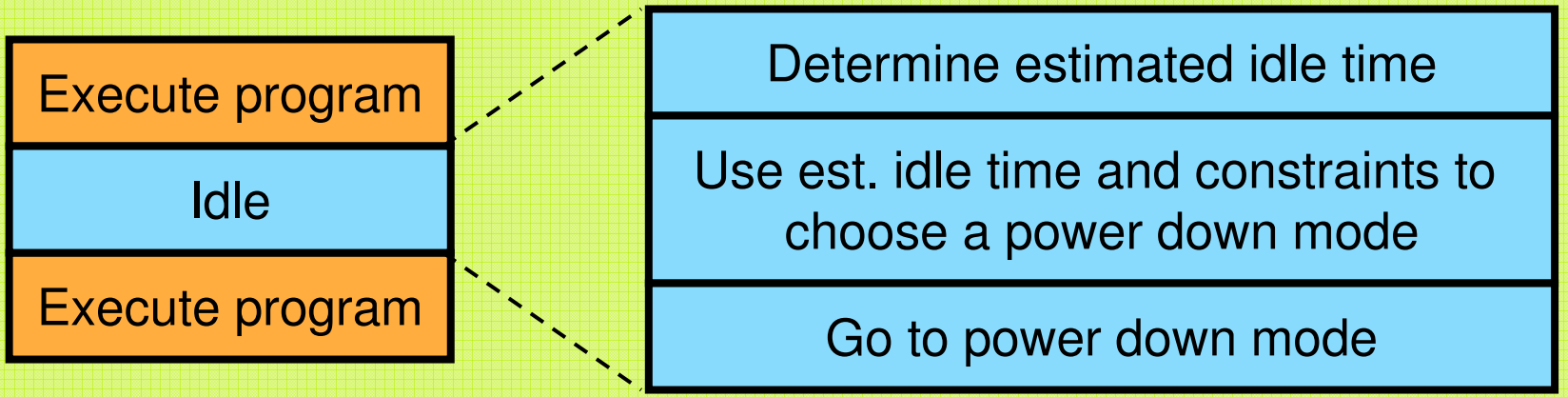


CPUidle in a nutshell



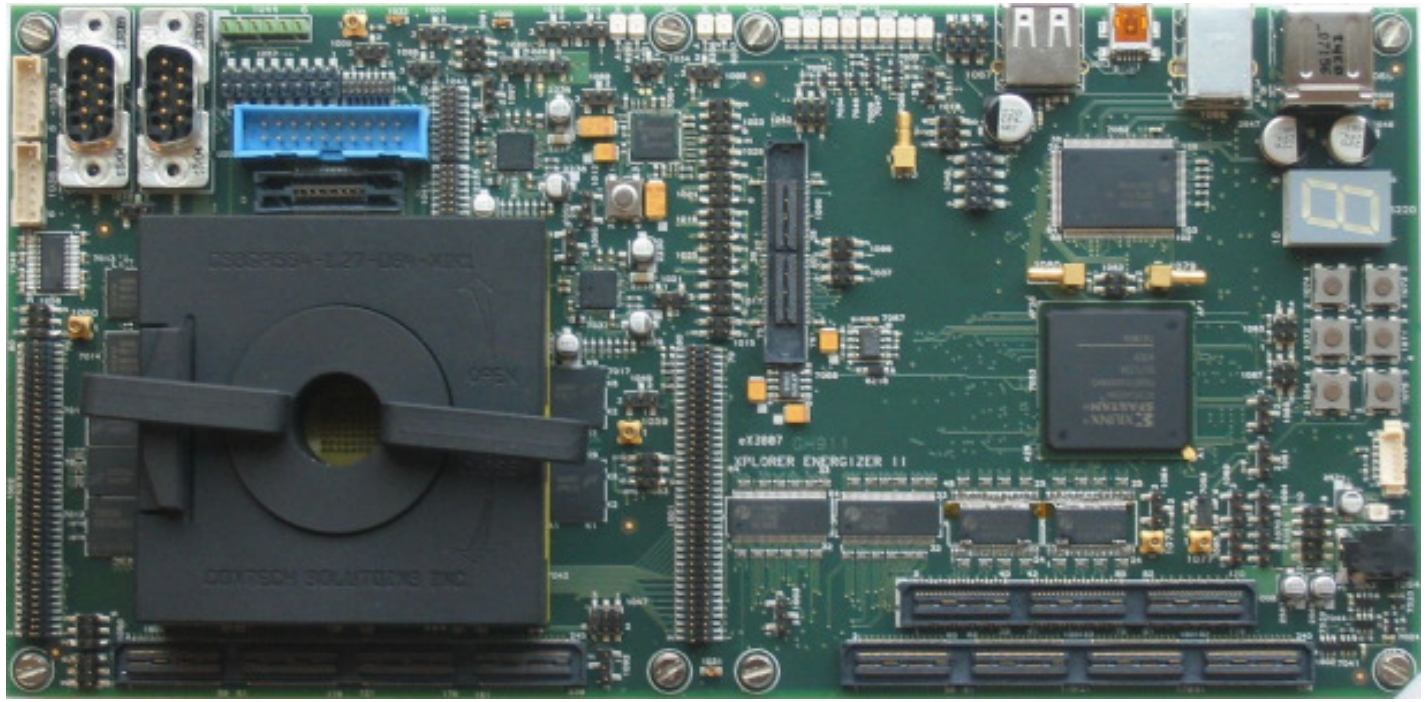
Old way

With CPUidle

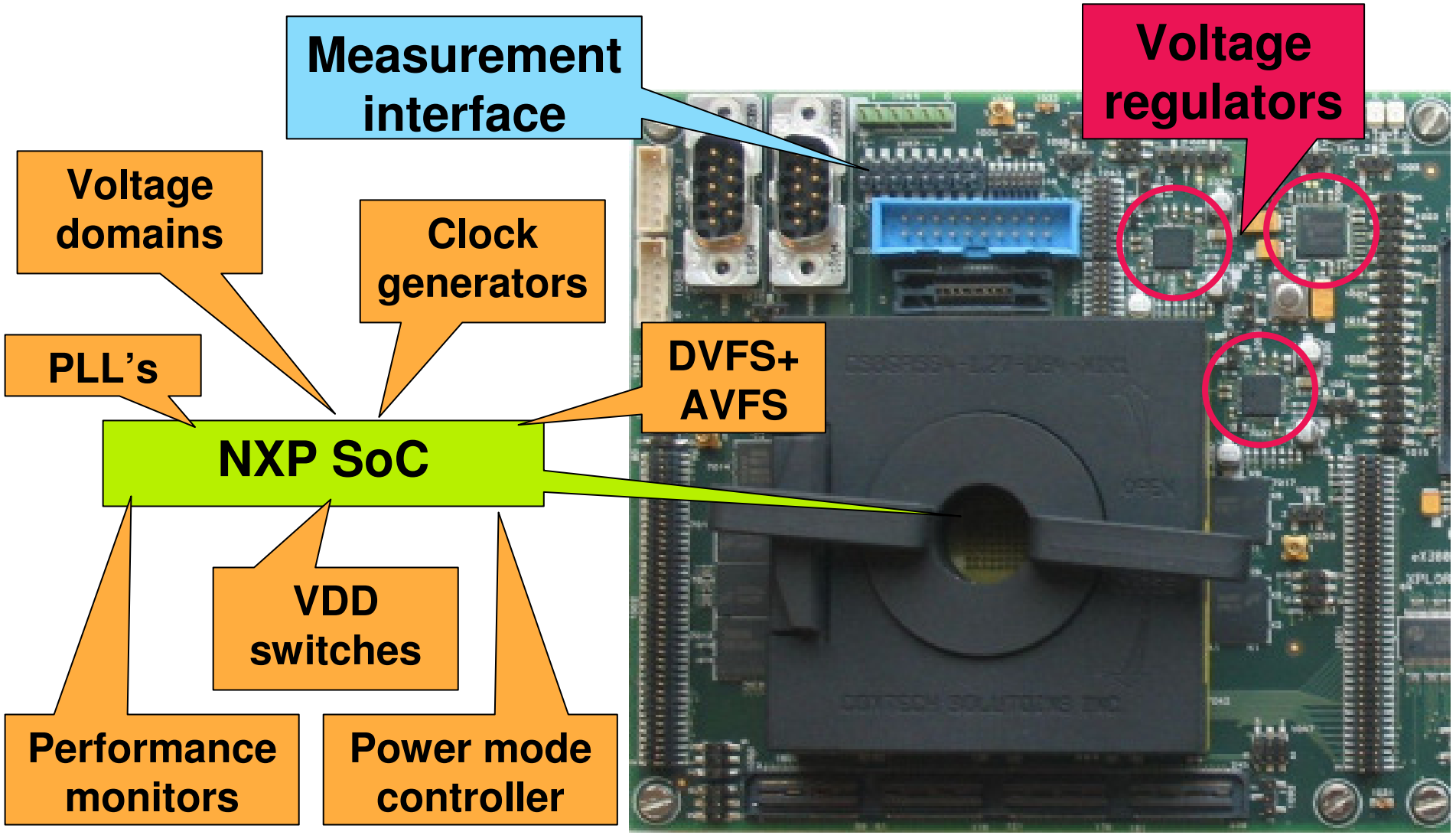


NXP ARM1176 platform

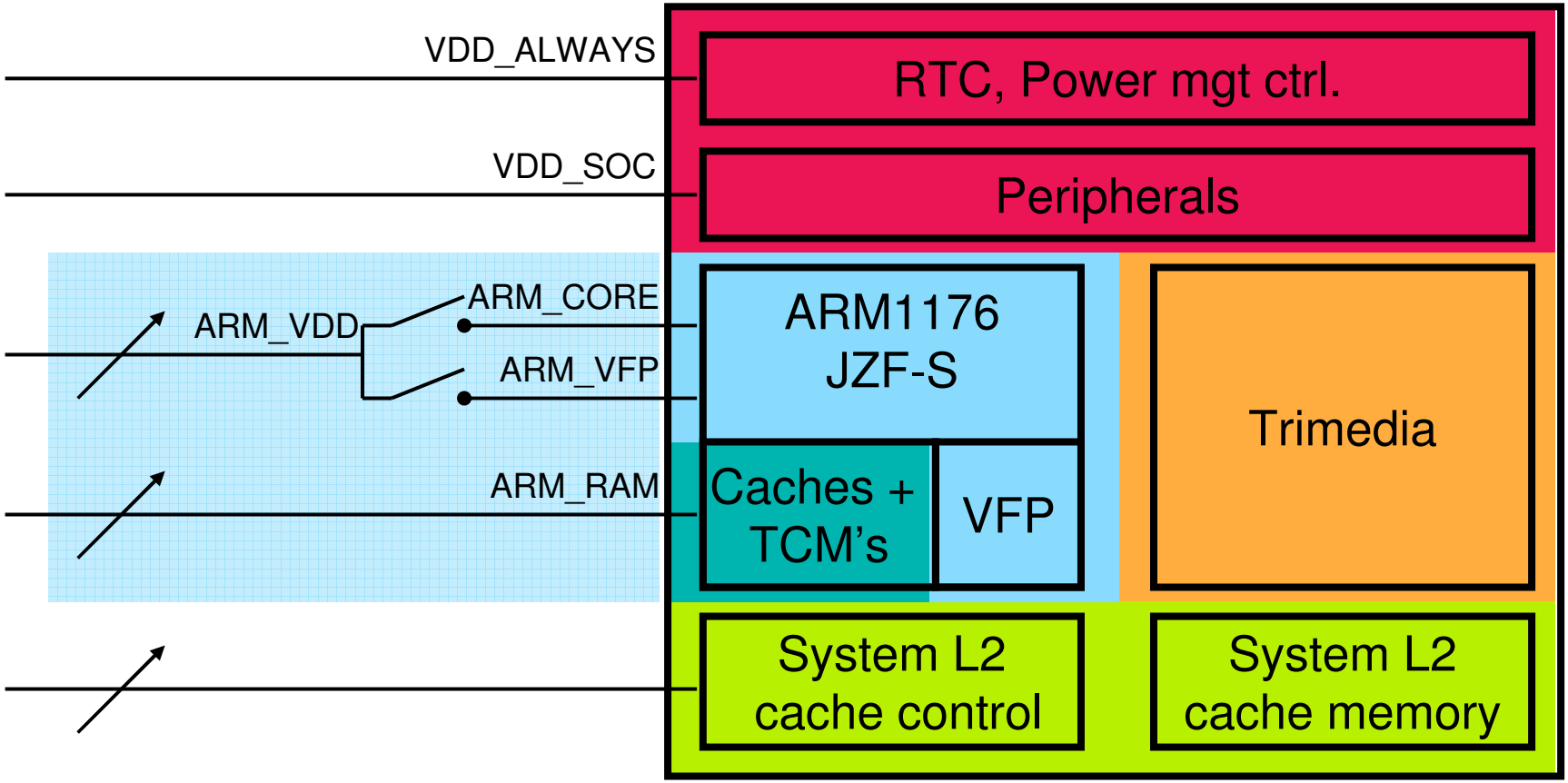
- CMOS 065 SoC with:
  - ARM1176, 32K I&D$, 4K TCM
  - TriMedia (DSP)
  - System L2 cache, 768KB
- Memory
  - 128MB LPDDR, 64MB Strata Flash, 32MB PSRAM, 256 MB NAND Flash



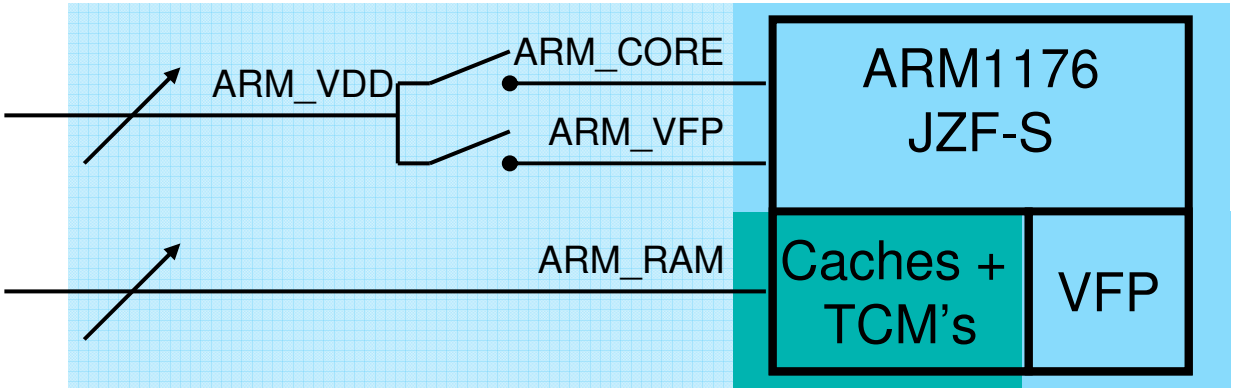
NXP ARM1176 platform: power management



NXP ARM1176 platform: Voltage domains

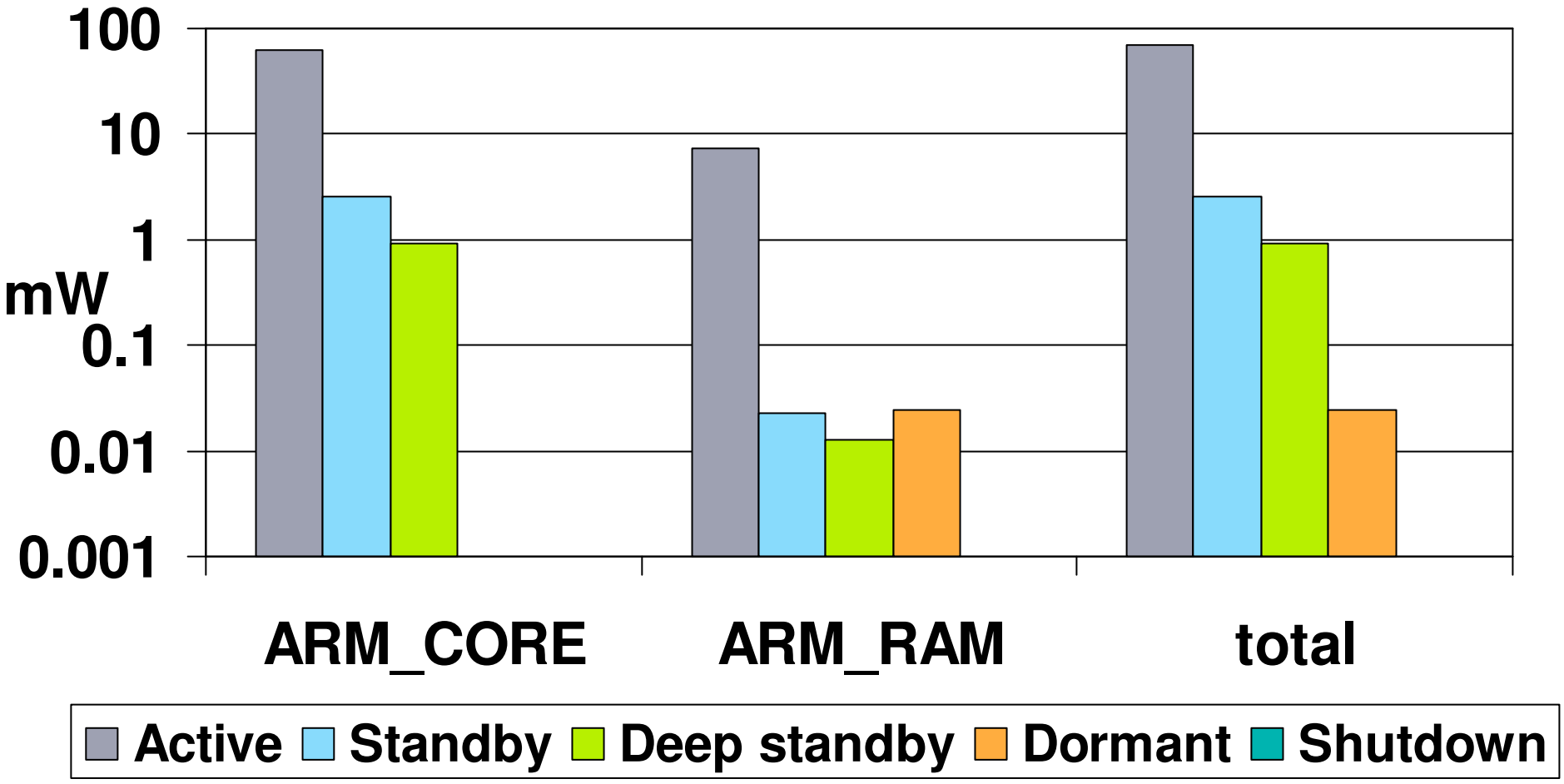


ARM11 power modes



	ARM core	ARM VFP	ARM RAM
ACTIVE	ARM_VDD	ARM_VDD / 0	ARM_RAM_VDD
STANDBY	ARM_VDD	ARM_VDD / 0	ARM_RAM_VDD
DEEP STANDBY	Vretention	Vretention / 0	Vretention
DORMANT	0	0	Vretention
SHUTDOWN	0	0	0

Power dissipation of ARM11 power modes



Entering and exiting power modes

Save status if required

Sleep event

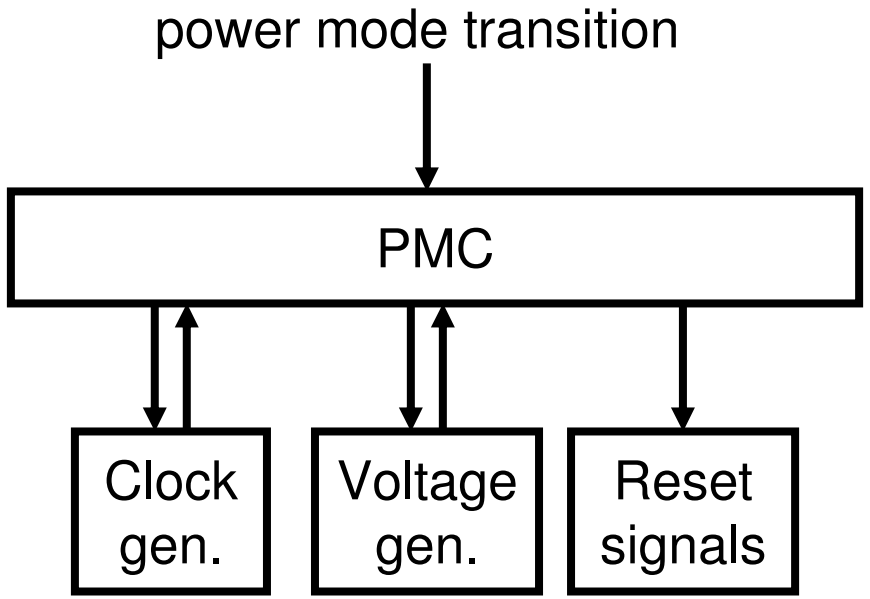
PMC does power mode transition

Wakeup event

Restore status

Power Management Controller (PMC)

- Controls power modes by controlling related components in a defined sequence.
- Listens to sleep and wakeup conditions to change operating modes.



Deep standby mode

- Pretty straight-forward mode transition:
  - You only need to program the PMC, for the rest it's fully transparent to the software.

Program PMC and voltage regulators for mode transition

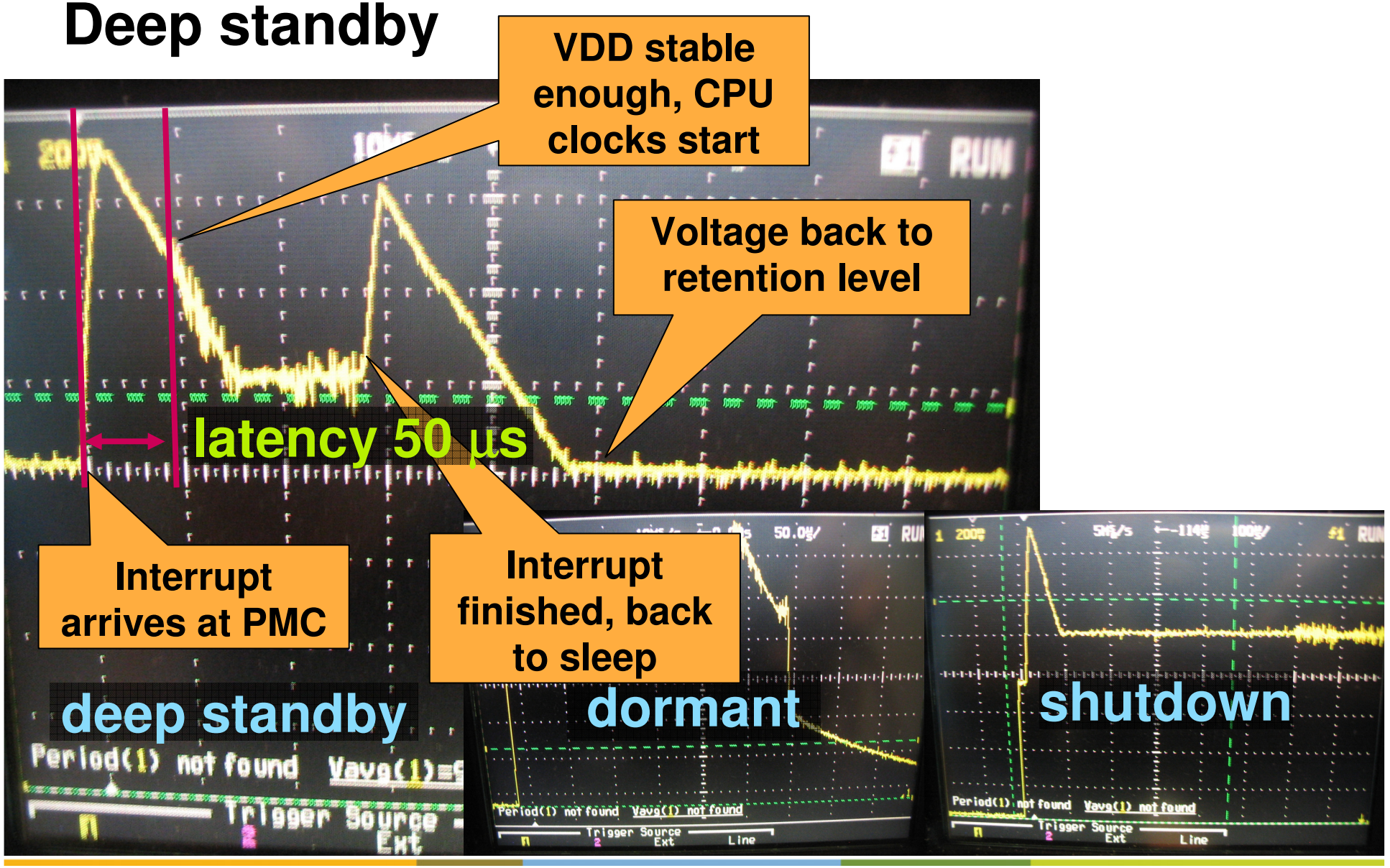
Execute WFI

PMC lowers all voltages to retention level (800mV) and stops relevant clocks

Interrupt wakes PMC up

PMC brings voltages back and starts ARM clock again

Deep standby



Entering dormant mode

- Disable all interrupts (and make sure exceptions cannot happen anymore)
- Program the PMC
  - set sleep trigger (Wait For Interrupt instruction)
  - set wakeup conditions (IRQ / FIQ)
  - set next power mode for all components: ARM_CORE, ARM_VFP, ARM_RAM
- Program the next voltage in the voltage regulators; this transition will be triggered by the PMC

Program PMC and voltage regulators for mode transition

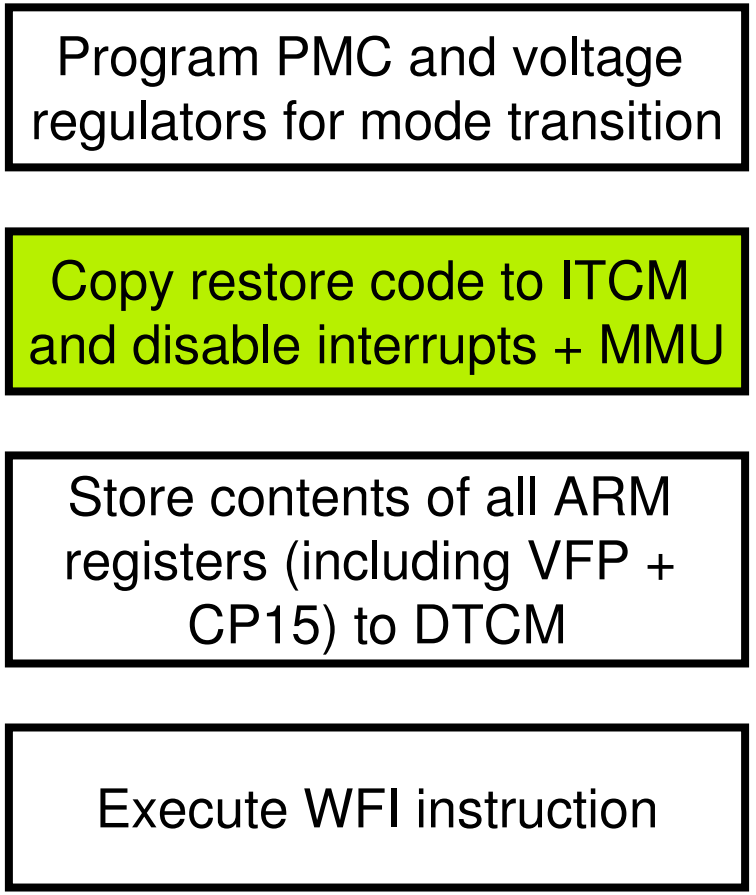
Copy restore code to ITCM and disable interrupts + MMU

Store contents of all ARM registers (including VFP + CP15) to DTCM

Execute WFI instruction

Entering dormant mode

- While we are storing all ARM registers, we are entering unexpected modes for the kernel
  - Should we receive an interrupt before power down, we need to go to restore code instead of interrupt handler!!
- Therefore, after copy of ITCM code, the MMU is turned off and the ARM is configured for low exception vectors.



Entering dormant mode

- Save all ARM registers
  - Be careful!! Some registers of the register file exist multiple times (for different modes: svc, int, fiq, abort, user)
- Save floating point regs
- Save CP15 regs
  - MMU status and more

Program PMC and voltage regulators for mode transition

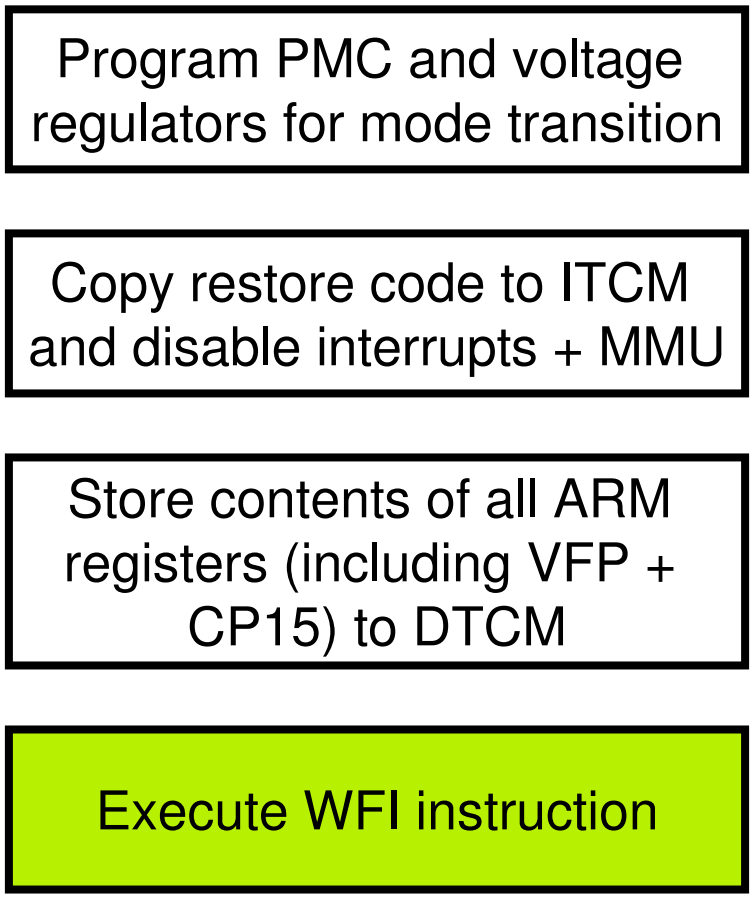
Copy restore code to ITCM and disable interrupts + MMU

Store contents of all ARM registers (including VFP + CP15) to DTCM

Execute WFI instruction

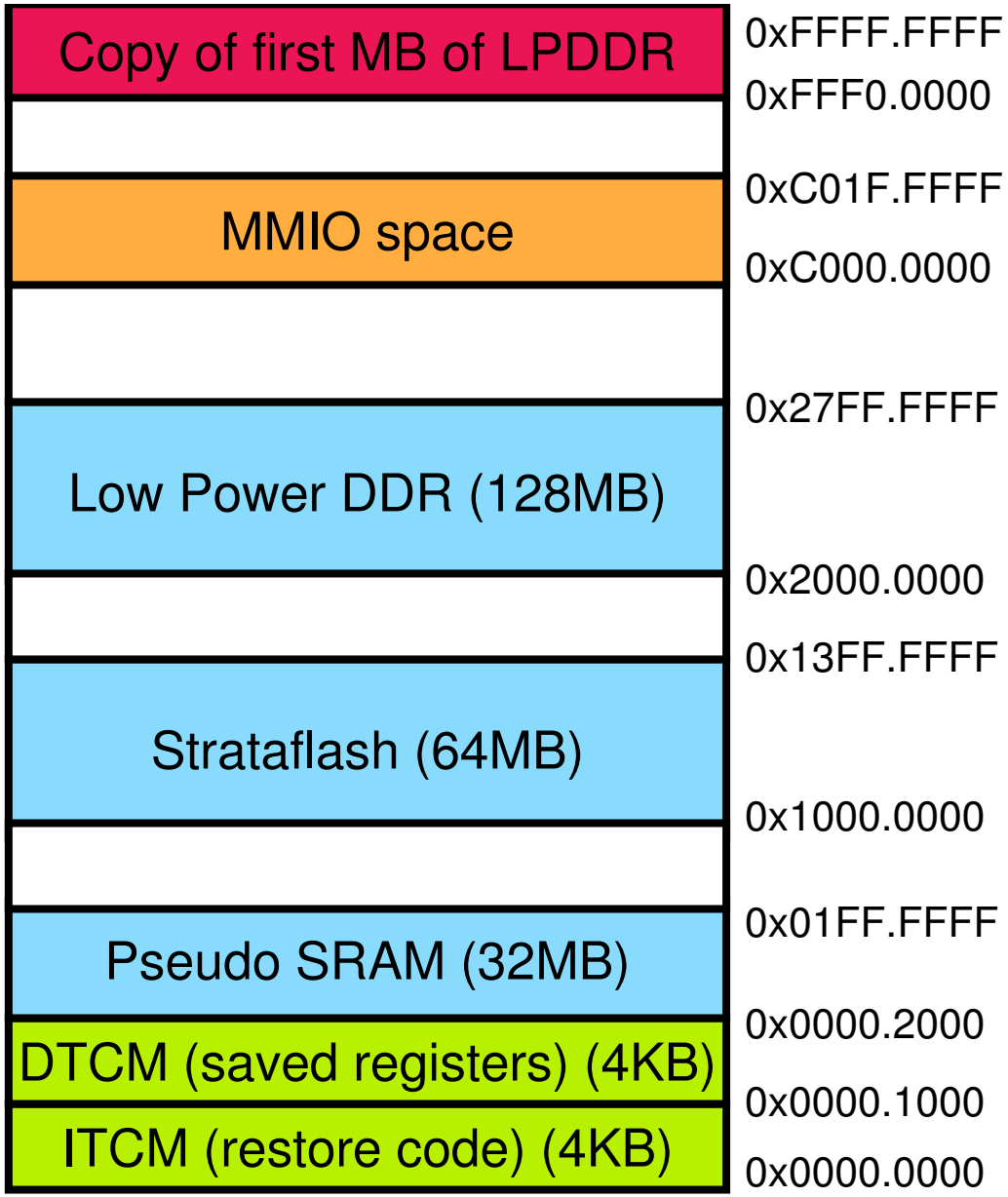
Entering dormant mode

- Enable interrupts again
- Execute WFI to trigger sleep



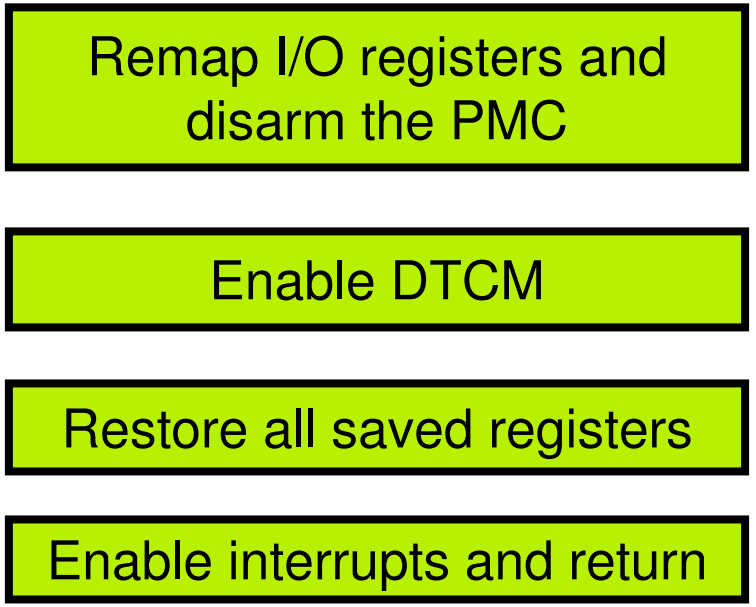
Memory map

- Three relevant signals:
  - INITRAM: initializes ITCM at 0x0000.0000 at boot-time
  - VINITHI: determines whether vectors are low or high at boot-time
  - BOOT_SMC: remaps first MB of LP-DDR to 0xFFF0.0000
- DTCM and ITCM are mapped over existing virtual memory map. So, needed to exclude these two pages for regular use!

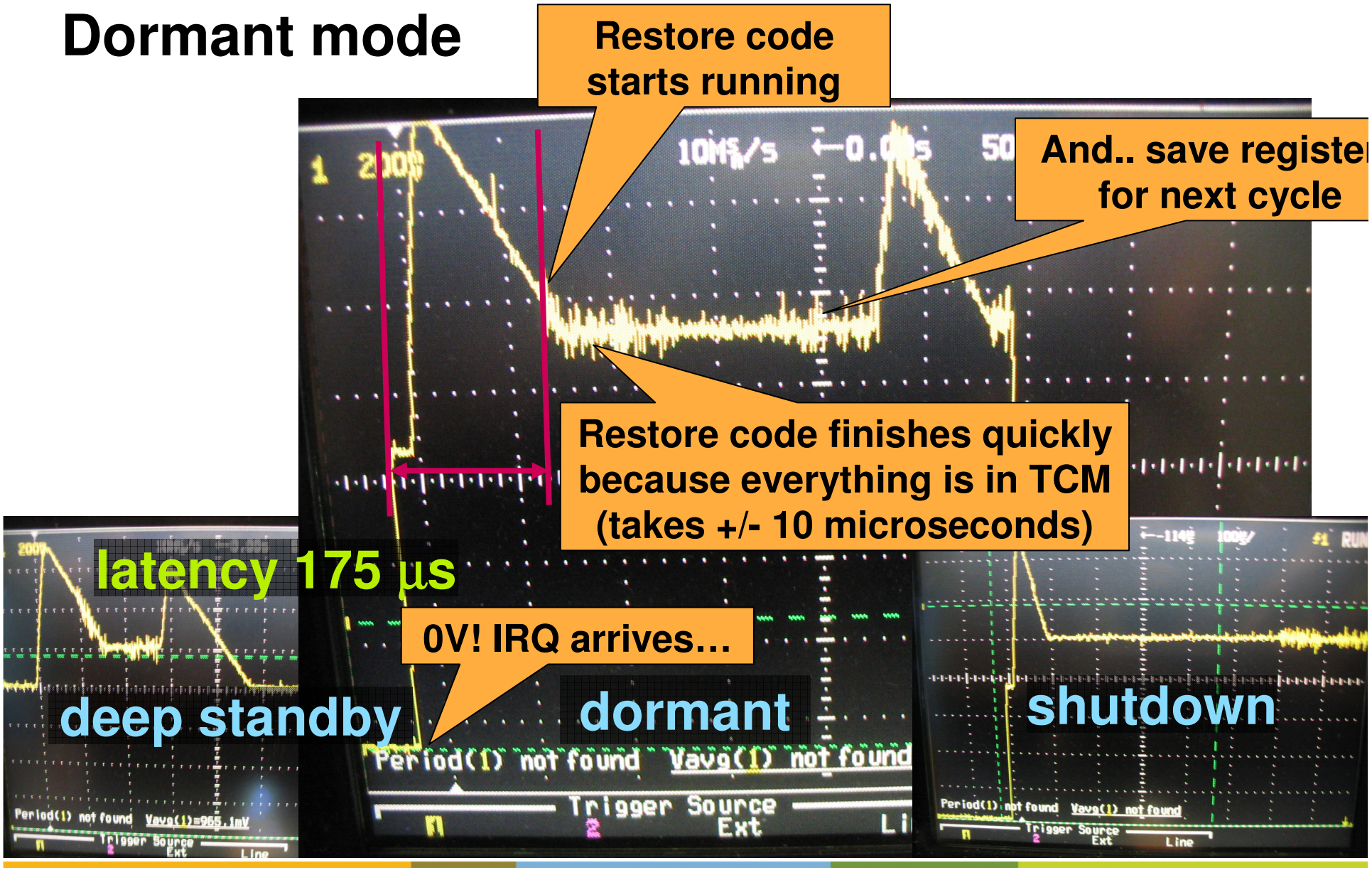


Waking up from dormant mode

- In the event of an interrupt before the WFI instruction, the PMC is still armed;
  - it could go to dormant next time when we do a WFI!
  - so we need to disarm it

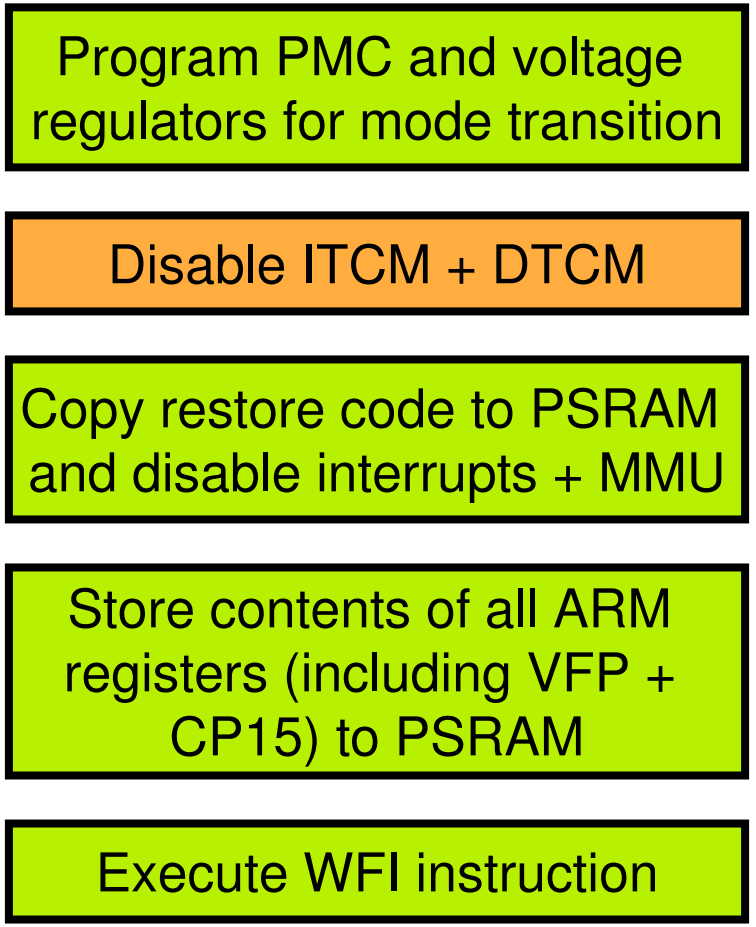


Dormant mode



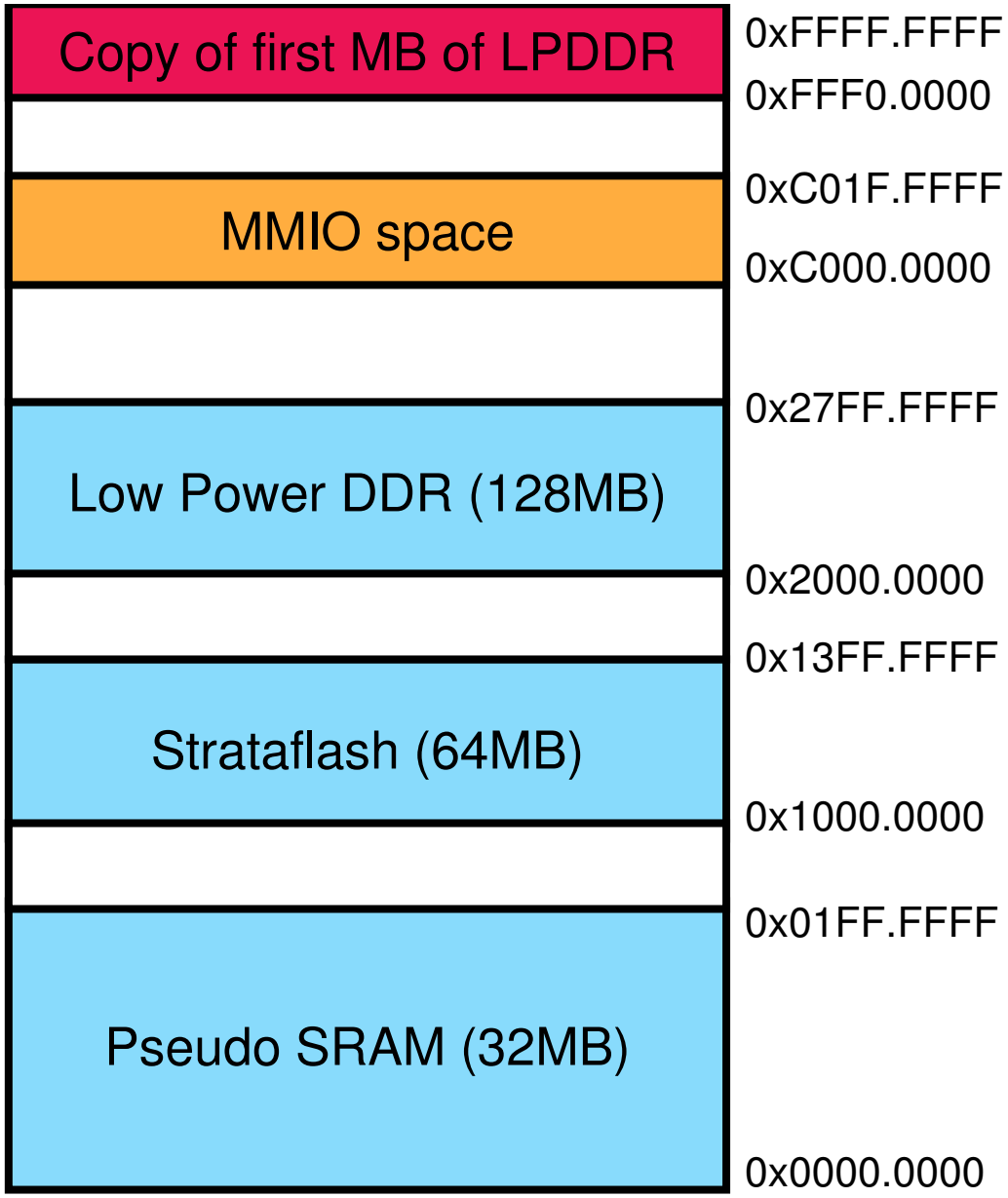
Entering + exiting shutdown mode

- Almost same procedure as with dormant, but state is saved to Pseudo SRAM instead of ITCM / DTCM.

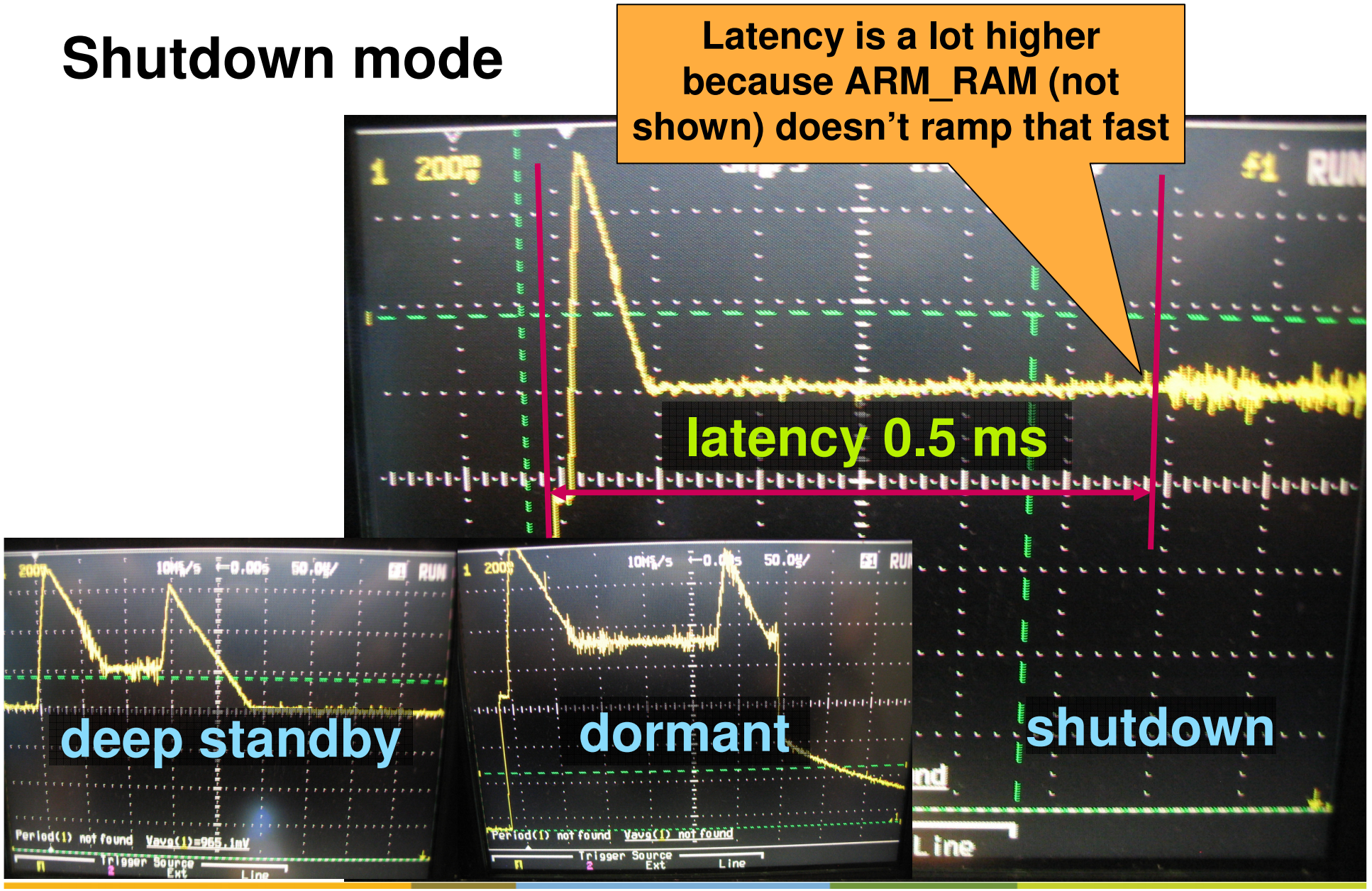


Memory map

- Three relevant signals: INITRAM, VINITHI and BOOT_SMC.
  - BOOT_SMC didn't work :-(
- So needed to use Pseudo SRAM for restore code --> slow....
- Reset vector is at 0x0000.0000, so we needed to swap flash & PSRAM
- Boot from flash is accomplished by writing a jump instruction in PSRAM by bootscript.

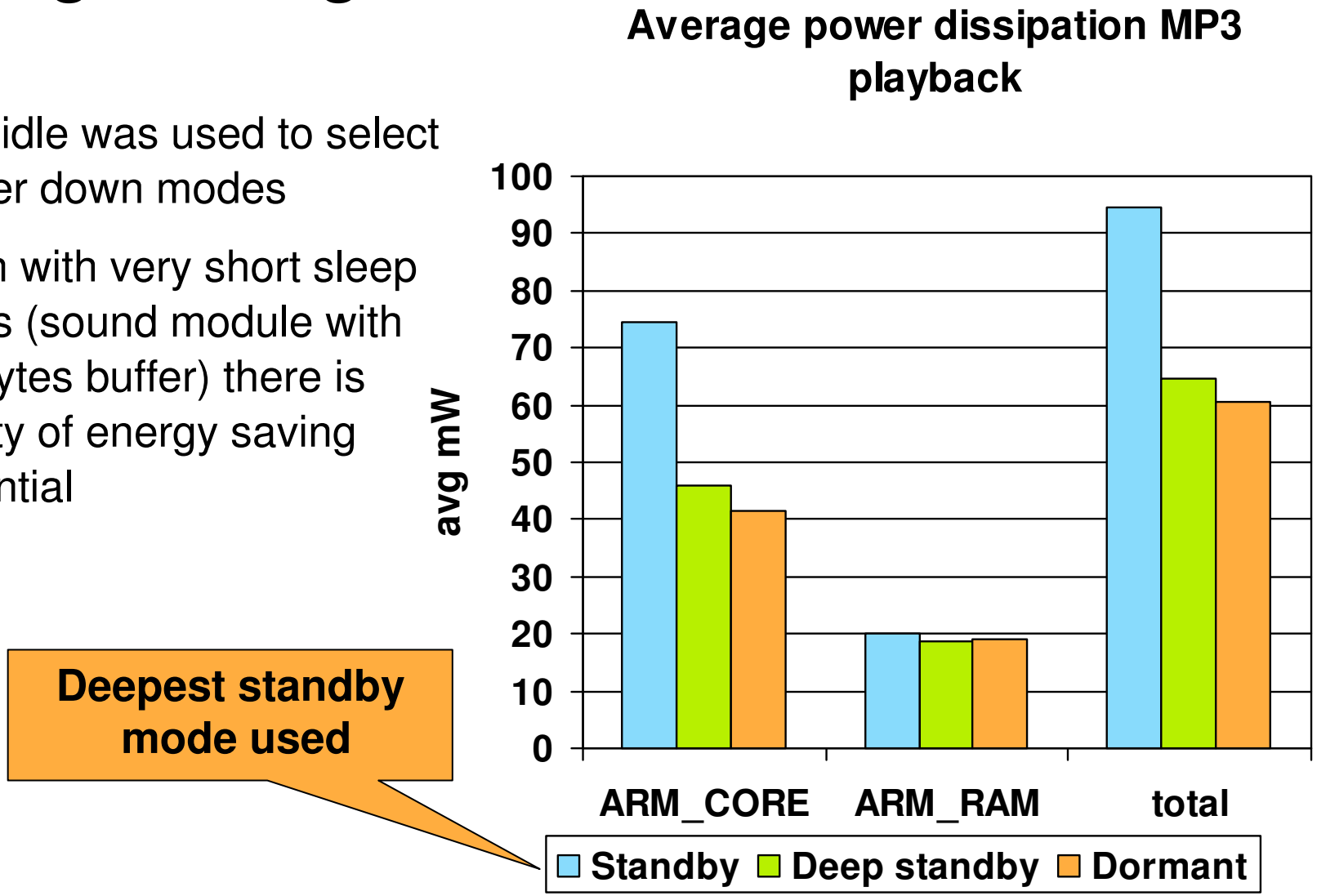


Shutdown mode



Putting it all together

- CPUidle was used to select power down modes
- Even with very short sleep times (sound module with 32 bytes buffer) there is plenty of energy saving potential



Encountered problems and future work

- Getting an interrupt after saving the state, but before WFI:
  - PMC is still armed, and on next ‘regular’ idle (standby) it will go to dormant/shutdown.
- Mysterious wakeups at 2.1 second intervals (don’t show up in powertop)
  - Turns out that 2.1 seconds is exactly 2^31 nanoseconds, so we moved to 64-bit integers for clock events.
- CPUidle is made with laptops in mind
  - Power dissipation for various idle states is measured in mW. This is not accurate enough for our platform.
- Some CPUfreq governors made the ARM wake up periodically
  - Solved in 2.6.24 by deferrable workqueues
  - A lot of drivers however still do periodic wakeups (USB stack: 4x a second)

Encountered problems and future work

- Latency framework is virtually unused
  - For instance our sound driver…
- Group timers, to decrease amount of wake-ups
  - Supported in GNOME (glib), but not in glibc, and not synchronized with the periodic kernel tasks.
- Disable the ARM_VFP voltage (through the power switch), and only enable it at the first unknown instruction exception. Furthermore keep track of which processes use the VFP, so that it can be enabled and disabled when switching tasks.

Conclusion

- A lot of power management kernel infrastructure exists:
  - not always tuned towards embedded use...
  - not everything is integrated...
- CPUidle is a nice way to handle ARM11 power modes